Predicting MOCS Scores Using Infant Kinematics Data

Reza Torbati, Andrew Fagg, Thubi Kolobe

May 2022

1 Introduction

The goal of this study is to measure how well infants are learning using the Self-Initiated Prone Progression Crawler V3 (SIPPC3) robotic system in an automated and reliable method. The SIPPC3 is an integration of robotics and sensor technologies designed to help infants who are at risk of cerebral palsy learn to crawl [2]. The SIPPC3 consists of a robot and an operator's laptop to control the robot [1]. An infant is placed in the robot in a prone position and the robot can then detect forces in the horizontal plane, presumably from the infant attempting to crawl, and move the robot in the appropriate direction [1]. The objective of the SIPPC3 is to teach the infant how to crawl by using reinforcement learning and error based learning.

Figure 1: Image of SIPPC3. *Photo credit* Sooner Magazine/Hugh Scott

The Movement Observation Coding Scheme (MOCS) was developed to quantitatively collect data about the strategies the infants use to move the SIPPC and measure how well they are learning [2]. The MOCS consists of 42 different measurements that fall under 4 different subscales: Subscale 1: Posture and Support; Subscale 2: Exploratory Selection and Progression; Subscale 3: Mastery of Propulsion; and Subscale 4: Socio-emotional Responses [2]. To generate these scores, each trial is videotaped and a researcher must watch the video after the trial to score each of the 42 different measurements [3]. This is a labor intensive process and may take up to 45 minutes to fully score a 5 minute trial. Additionally, there can be inconsistencies between different researchers scoring the videos [3]. The MOCS scores were developed so that they would be consistent across researchers but there is still some subjectivity involved. If experienced researchers were used, Rule (2010) showed that the Intraclass Correlation Coefficient between researchers ranged from .745 for subscale 4 to .868 for subscale 3 [3].

In this study, I am focusing on predicting subscale 3: Mastery of Propulsion. The Mastery of Propulsion subscale aims to capture the ranges of responses that describe how the infants' skill in locomoting using the SIPPC is developing [3]. To accomplish this, the Mastery of propulsion subscale is broken into 6 different categories. Every 30 seconds, the researcher assigns the infant into a single category based on how the infant engaged with the SIPPC in those 30 seconds. Table 1 summarizes the different categories.

Action	Category	Score
$>$ 5 moves consecutively	33	4
2-4 moves consecutively	32	3
Up to 1 move consecutively	31	2
Attempt to move or reach for toy without moving	30	1
Playing without attempting to move	29	-1
Not engaged	28	-2

Table 1: Each category the researcher can assign the infant for the Mastery of Propulsion subscale and that category's value [3]

At the end of the trial, the infant's final Mastery of Propulsion score is calculated by giving the infant -2 points for every 30 second interval that it was put in category 28, -1 points for every interval it was put in category 29, 1 point every time it was put in category 30, 2 points every time it was put in category 31, 3 points every time it was put in category 32, and 4 points every time it was put in category 33. The final score is the score that is then primarily analyzed to find how well the different infants are learning to use the SIPPC from trial to trial.

In addition to data describing the movement of the SIPPC robot and how it is responding to the infant, a motion capture suit worn by the infant captures the position of the trunk and limbs. Because the Mastery of Propulsion score

is generated based on how well the baby is interacting with the SIPPC, we hypothesize that this score can be estimated given the the robot and kinematic data.

2 Experiment Setup

The data I used to train a model consisted of 239 scored trials from 27 different infants, six of whom were at risk of cerebral palsy and 21 of whom where typically developing. My goal was to predict which category the infant fell under for each 30 second interval. Each trial was five minutes long, so I had 10 examples from each trial to work with. All but two of the 30 second intervals were valid so I had a total of 2,388 different examples for training, validating and testing the models.

Category	Examples	Percent of Total Examples
28	255	10.6
29	78	3.24
30	239	9.93
31	890	37.0
32	778	32.3
33	168	6.98

Table 2: Class Distribution Among My Examples

As is shown in Table 2, there is an imbalance in the class distribution. This is especially true for category 29. Because this category is so rare, we decided to remove all 30 second intervals where 29 was the given score. This left me with a total of 2310 examples.

For the inputs for the model, I used the recorded time series data from the SIPPC. Specifically I used the following features:

- The infant's x, y and z coordinates for its back
- The infant's x, y and z coordinates for its left and right ankles
- The infant's x, y and z coordinates for its left and right elbows
- The infant's x, y and z coordinates for its left and right feet
- The infant's x, y and z coordinates for its left and right knees
- The infant's x, y and z coordinates for its left and right shoulders
- The infant's x, y and z coordinates for its left and right wrists
- The x, y, and θ values for the Cartesian velocity of the robot
- 6 dimensions from the onboard force-torque sensor that describes the forces and torques imposed by the infant including:

  - dynamic effects, such as changes in the infant's body configuration, and
  - static effects, such as contact with the floor.

- A one-hot encoding that describes the discrete response of the robot to the infant action (no movement, power-steering or suit steering).
- A one-hot encoding that describes if the SIPPC is producing a discrete response to the left, right, forward or backward.

This added up to 51 different features. Each of these features were sampled at 50Hz, so there was a total of 76500 individual values used to predict the class of each of the 30 second intervals.

3 The Model

3.1 Overview

The models that I created to predict the outputs for the 30 second intervals use one-dimensional convolutional neural networks. Convolutional neural networks create filters that can summarize segments of data with more meaningful features. Examples of filters include looking for when the right arm makes a specific motion. This would require the model to look at the information from the infant's right wrist and elbow to find if this specific motion was performed over a specific amount of time. If the motion was performed, then it may be more likely that the infant is attempting to move the SIPPC. In turn, this observation might suggest that the infant should receive a higher score. Using convolutional neural networks, it is possible for the model to automatically learn dozens of filters that can then be cascaded into additional layers with even more filters. Eventually, the model can significantly reduce the number of time points that it is looking at without losing key information.

Parameters for a deep network are chosen through a gradient-descent search process in which parameters are adjusted incrementally so as to minimize some loss function. In this case, the loss is defined as the categorical cross-entropy between the estimated class probabilities and the true class labels. It is a measure of dissimilarity between the estimated probabilities and the true class label. In this case, the true label is the category that the infant was put in by the researcher and the model's output is how likely it thinks that the researcher put the infant in each category. The category with the highest probability is what is then selected as the model's prediction.

3.2 Bayesian Optimization

Because the model is so complex, there are dozens of different hyper-parameters that needs to be chosen. Hyper-parameters include the number of layers of

cascading filters are used, how many filters are used for each layer, the model's learning rate and much more. To find the best combination of hyper-parameters, I used Keras's Bayesian Optimizer. This uses Bayesian statistics to attempt to statistically find which combination of hyper-parameters yields the best results.

My setup for the Bayesian Optimizer was to use a naive approach where I gave the optimizer full control over all possible hyper-parameters. This was a total of 73 different hyper-parameters. The optimizer was then executed for 317 iterations. I then took the best set of hyper-parameters for the model's architecture, as measured by the validation set performance. The full model architecture that was created by the optimizer is shown in Section A

3.3 Cross-Validation

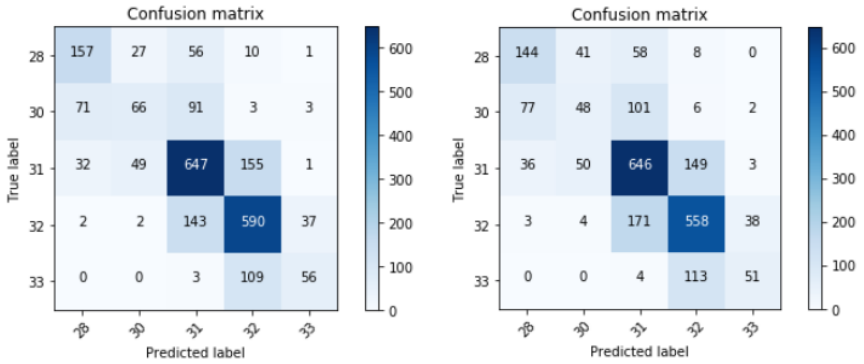
I used cross validation to ensure that the approach was generalizable. To accomplish this, I split my data into six different folds, each of approximately equal size and class distribution. Each fold had all of the trials for one of the infants at risk of cerebral palsy and a mix of the trials with typically developing infants. I then created six rotations, where each rotation used four of the folds as training data, one fold as validation data, and one fold as test data. The training data is used to actually train the model. The validation data is used to stop the model when it starts to overfit on the training data. The test data is then used to see how the model performs on data that is completely unseen by the model during the training process. Because there were six separate rotations, six different models were trained using the same architecture to ensure that the final model architecture is generalizable to unseen data.

4 Results

Each rotation created by this model performed very well. The Bayesian Optimizer was tuned primarily using rotation three but every rotation showed statistical significance when predicting both the validation data and the test data. Figure 2 shows the confusion matrices for the validation and test data across the set of six models. High counts along the diagonal indicate a high prediction accuracy. In addition, most prediction errors correspond to a class difference of one.

As previously mentioned, the 30 second intervals that were scored as category 29 were removed due to how rare they were. The average validation accuracy across all six rotations is 0.656 and the average test accuracy is 0.626. The average accuracy for a model that randomly predicts the labels with the same distribution as the true labels is .306 with a standard deviation of .003.

Additionally, the models showed promising results when predicting the full five minute Mastery of Propulsion scores. When comparing the predicted scores with the true scores, the intervals that were given the class 29 were removed. To compensate for the loss of one of the 30 second intervals from the 5 minute trial, the final score was scaled according to how many valid intervals there were. For



A B

Figure 2: Confusion matrices across all models for the validation (A) and test (B) data, respectively. The number in each cell is the count for the combination of true and predicted label, with the diagonal corresponding to a match between the two.

example, if a trial had two 30 second intervals that were categorized as class 29, the final Mastery of Propulsion score was calculated as the score from the other eight intervals multiplied by 10/8.

Across all six models, the mean absolute prediction error was 4.14 for the validation data. Score predictions correlated significantly with the given scores ($p < 10^{-63}$), with the fraction of variance accounted of .665 and R^2 of .696. Additionally, the Intraclass Correlation Coefficient (ICC) was .835. In the original reliability experiments conducted on the MOCS scores, the ICC between different researchers was found to be 0.799 when a novice was included and and .868 when there were only experienced researchers included [3]. The target ICC set by the original researcher was .85 so the current version of the models are already very close to meeting that threshold.

Additionally, many of these results held up on the test data, which was data that was completely unseen by the models when they were being trained. For the test data, the mean absolute error was 4.50. In addition, predictions correlated significantly ($p < 10^{-59}$) with the given scores, with a fraction of variance accounted for of .640, an R^2 of .674 and an ICC of 0.822.

This shows that the model was able to generalize very well to unseen data. However, as can be seen in Figure 3, the models do not perform as well when the infant obtains a low score. This can be explained by two factors. First, the model does not have nearly as many training examples where the true label is 28 or 30, as it does for labels 31 and 32. Second, if the model predicts 31 instead of 32, the final Mastery of Propulsion score is only changed by 1. However, if the model predicts 30 instead of 28, the final Mastery of Propulsion score is changed by 3. This leads to an inherent instability in the lower scores. This is further compounded by the fact that the model's loss is categorical cross entropy. This means that the model weights each category's importance based on its

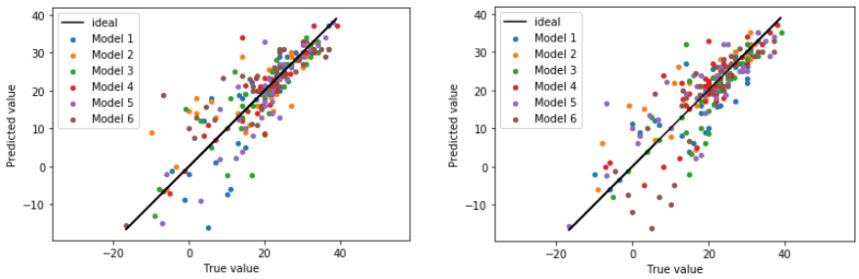


Figure 3: Scatter plot of the predicted trial scores compared to the true scores for the validation (A) and test data (B) sets. The solid line indicates the ideal relationship between the scores. Dot color indicates the model that generated the particular prediction.

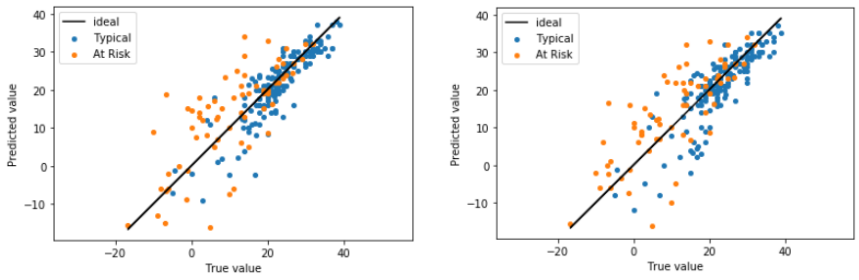


Figure 4: Scatter plot of the predicted trial scores compared to the true scores for the validation (A) and test data (B) sets. Dot color indicates subjects that are typically developing (blue) and at risk (orange).

proportion of the training examples so the model cares more about predicting 31 and 32 correctly than it does about 28.

This problem can be further seen when comparing how the model predicts the final Mastery of Propulsion scores for infants at risk of Cerebral Palsy, as compared with typically developing infants (Figure 4). The R^2 value drops from 0.777 for typically developing infants to .560 for infants who are at risk for the validation data and drops from .714 for typically developing infants to .589 for at risk infants for the test data. Additionally, the ICC for the test data drops from .822 for typically developing infants to .739 for at risk infants. This can be primarily explained by the fact that at risk infants tend to score much lower than typically developing infants and the model is much more accurate for infants who score higher. However, predicted scores for at risk infants are still significantly correlated with the given scores ($p < 10^{-13}$).

5 Future Work

There are a few things that can be done that may significantly improve this model. First, more data are required. The reason why I stopped tuning the hyper-parameters after 317 trials is because the optimizer was starting to overfit to the validation data. Currently, less than half of the available data are used for training, validating and testing of the models.

Additionally, the Bayesian optimizer should be refined. The naive approach of giving it total control of all hyper-parameters created a model that was better than what I could train when choosing the hyper-parameters by hand. However, it only had 317 trials to tune over 70 hyper-parameters. If there were fewer hyper-parameters for it to tune, it is possible that a more appropriate set of the hyper-parameters could have been discovered.

Finally, the models may be improved if the loss was changed from categorical cross entropy to a loss function that takes class ordinality into account. Likewise, including class weights in the loss function that emphasize classes 28 and 30 over the other classes could improve prediction of these classes in the validation and test data sets. In turn, this could improve trial scores for the lower-score cases.

A Appendix

A.1 Full Model Used

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 1500, 51)]        0
_________________________________________________________________
AdvPool (AveragePooling1D)   (None, 750, 51)           0
_________________________________________________________________
C00 (Conv1D)                 (None, 748, 71)           10934
_________________________________________________________________
SpatialDrop00 (SpatialDropou (None, 748, 71)           0
_________________________________________________________________
C01 (Conv1D)                 (None, 748, 46)           3312
_________________________________________________________________
Max01 (MaxPooling1D)         (None, 250, 46)           0
_________________________________________________________________
SpatialDrop01 (SpatialDropou (None, 250, 46)           0
_________________________________________________________________
C02 (Conv1D)                 (None, 250, 71)           3337
_________________________________________________________________
SpatialDrop02 (SpatialDropou (None, 250, 71)           0
_________________________________________________________________
C03 (Conv1D)                 (None, 244, 36)           17928
```

Max03 (MaxPooling1D)	(None, 122, 36)	0
SpatialDrop03 (SpatialDropou	(None, 122, 36)	0
C04 (Conv1D)	(None, 116, 36)	9108
Max04 (MaxPooling1D)	(None, 58, 36)	0
SpatialDrop04 (SpatialDropou	(None, 58, 36)	0
C05 (Conv1D)	(None, 58, 41)	1517
Max05 (MaxPooling1D)	(None, 20, 41)	0
SpatialDrop05 (SpatialDropou	(None, 20, 41)	0
C06 (Conv1D)	(None, 20, 61)	2562
SpatialDrop06 (SpatialDropou	(None, 20, 61)	0
C07 (Conv1D)	(None, 20, 16)	992
SpatialDrop07 (SpatialDropou	(None, 20, 16)	0
C08 (Conv1D)	(None, 16, 71)	5751
Max08 (MaxPooling1D)	(None, 8, 71)	0
SpatialDrop08 (SpatialDropou	(None, 8, 71)	0
flatten_1 (Flatten)	(None, 568)	0
D00 (Dense)	(None, 90)	51210
Drop00 (Dropout)	(None, 90)	0
D01 (Dense)	(None, 48)	4368
Drop01 (Dropout)	(None, 48)	0
D02 (Dense)	(None, 48)	2352
Drop02 (Dropout)	(None, 48)	0
D03 (Dense)	(None, 36)	1764

```
_________________________________________________________________
Drop03 (Dropout)             (None, 36)                0
_________________________________________________________________
output (Dense)               (None, 5)                 185
=================================================================
Total params: 115,320
Trainable params: 115,320
Non-trainable params: 0
```

References

[1] Mustafa A Ghazi et al. "Novel assistive device for teaching crawling skills to infants". In: *Field and service robotics*. Springer. 2016, pp. 593–605.

[2] Thubi HA Kolobe and Andrew H Fagg. "Robot reinforcement and error-based movement learning in infants with and without cerebral palsy". In: *Physical Therapy* 99.6 (2019), pp. 677–688.

[3] Barbara Ann Rule. "Motor strategies implemented by infants using the self-initiated prone progression crawler". MA thesis. University of Oklahoma, 2010.